

Draft tesi (latest major changes in orange)

Enrico Santi

June 27, 2022



UNIVERSITY
OF UDINE

Department of
Mathematics, Computer Science and Physics

BACHELOR THESIS IN
Computer Science

**Numerical bifurcation analysis of delay equations:
a user-friendly extension of MatCont's interface**

CANDIDATE

Enrico Santi

SUPERVISOR

Prof. Rossana Vermiglio

Co-SUPERVISOR

Dr. Davide Liessi

Contents

1	Introduction	2
2	Basics on dynamical systems	3
2.1	Dynamical systems	3
2.2	Ordinary differential equations	3
2.3	Delay differential equations	6
2.4	Stability and bifurcations	7
2.5	From DDEs back to ODEs	8
3	A brief introduction to MatCont	10
3.1	Overview of the current MatCont GUI	10
3.1.1	MatCont Main Window	10
3.1.2	Inserting a new system	11
3.1.3	Editing an existing system	13
3.2	The existing documentation	13
4	Extending the user interface of MatCont	14
4.1	MatCont's extended capabilities	14
4.2	Overview of the extended MatCont GUI	15
4.2.1	The DDE syntax	15
4.2.2	Inserting a new ODE or DDE system	16
4.2.3	Editing an ODE or DDE system	17
4.3	The complete legacy support	18
4.4	A deeper view of the parsing process	21
4.5	A brief look at the development process	23
5	Conclusions	25
5.1	Future extensions	25

Introduction

This thesis mainly concerns the work carried out during the internship at the CD-Lab at the Department of Mathematics, Computer Science and Physics of the University of Udine. The internship has been focused on the development of an extension of the graphical user interface (GUI) of MatCont, a MATLAB software for continuation and bifurcation of ordinary differential equations (ODEs). The aim is to provide a user-friendly GUI to study dynamical systems described by delay differential equations (DDEs), which arise in many application fields, i.e. control theory, ecology and epidemiology. The thesis is organized as follows.

In Chapter 2 after a brief introduction on differential equations and dynamical systems we introduce the discretization approach, which is the theoretical basis of the MatCont extension, to approximate DDEs by a system of ODEs. Chapter 3 we describe the fundamental aspect of the current version of MatCont's GUI and how it works while in Chapter 4 we will focus on the new capabilities of the extended version of the extended version of MatCont, what are the new capabilities of this software and how the development has been carried out. Finally in Chapter 5 we conclude with some remarks on the future additional tools to implement to analyse the dynamics of renewal equations too.

The work done and described in Chapter 4 will also be presented at the *17th IFAC Workshop on Time Delay Systems (TDS)*, September 2022 in Montreal, Canada.

2

Basics on dynamical systems

2.1 Dynamical systems

Dynamical systems describe the evolution in time of phenomena and find application in several fields, from engineering to biology ([1, 2]). Mathematically a phenomenon is represented by a state variable and the dynamical system is defined as a triple $\{T, X, \phi^t\}$ where T is the time set, X is the state space¹, and $\{\phi^t\}_{t \in T}$ is the family of evolution operators $\phi^t : X \rightarrow X$ satisfying:

- $\phi^0 = id$, where id is the identity on X ,
- $\phi^{t+s} = \phi^t \circ \phi^s$.

A first major distinction is related to the time set T :

- when $T \subset \mathbb{N}$ we call them discrete dynamical systems,
- when T is a more general subset of \mathbb{R} , such as an interval, we call them continuous dynamical systems.

As described in [3] the relationship between discrete and continuous systems recalls the relationship between $\Delta x / \Delta t$ and dx/dt . In this thesis we will treat only continuous dynamical systems defined by ordinary differential equation (ODE) and delay equations, so from here we refer to them simply as dynamical systems.

2.2 Ordinary differential equations

We consider autonomous ordinary differential equations of the form:

$$y'(t) = F(y(t)), \quad t \geq 0 \tag{2.1}$$

where t is the independent variable, y the unknown function (so $y(t) \in \mathbb{R}^d$ is the state) and \mathbb{R}^d the state space with $d \geq 1$, and $F : A \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$. Solving a differential equation (when it is possible) means finding a set of functions satisfying the relationship on an interval. To obtain a unique

¹all the values describing the state the considered dynamical system can assume as the time progresses.

solution additional conditions are needed. Here we consider the following initial value problem (IVP):

$$\begin{cases} y'(t) = F(y(t)), & t \geq 0 \\ y(t_0) = y_0 \end{cases} . \quad (2.2)$$

Under suitable assumptions on the right hand side F , the IVP (2.2) has a unique solution $y(t) = \phi^t(y_0)$, for $t \in [0, t_f]$. Moreover it can be proven that the solution continuously depends on the initial value y_0 [4]. Note that the state space is \mathbb{R}^d so the equation (2.1) defines a finite-dimensional dynamical system.

A trajectory of a solution of an ODE, in particular for a dynamical system, consists of the set of the values of the unknown function in the interval and the values of the independent variable itself (i.e. $(t, y(t)) : t \in I$). Related to this concept, one of the first geometric visualizations for ODEs is the trajectory diagram. This diagram, in relation to a given ODE, presents on one of its axes the independent variable, and on the other the unknown function, for what concerns systems of ODEs components (unknown functions) is considered separately in different diagrams. What this diagram shows is a vector field (i.e. an assignment of a vector to each point in a given space), where at each point, the slope of the corresponding vector represents the value of the first derivative of the unknown function at that specific point. This becomes really clear when the definition of derivative is recalled, which can be informally described as a function that associates to each point of a differentiable function the angular coefficient of the tangent vector in that point, if it exists.

Example 1. The (continuous)² Malthus equation is defined as follows:

$$y'(t) = ry(t) \quad (2.3)$$

It is a linear autonomous first order ODE in which r represents a parameter, that can be solved analytically by separation of variables³.

The general solution of the equation is $\phi^t(y_0) = y_0 e^{rt-t_0}$ for $t \geq 0$ where y_0 is the initial value chosen for $y(t_0)$.

An equilibrium, a constant solution on the domain, for the equation (regardless the value of the parameter r) is obtained with the initial value $y_0 = 0$ at $t_0 = 0$. Plotting the solution of the relevant IVP (with different values of r) will always result in a straight line on the trajectory plane.

Example 2. The logistic model is described by the logistic equation:

$$y'(t) = ry(t)\left(1 - \frac{y(t)}{K}\right) \quad (2.4)$$

²its discrete counterpart is described by the recursive equation $y(t+1) = y(t) + ry(t)$

³A procedure for solving ODEs that can be rewritten as $y'(t) = a(t)b(y)$

This model describes the ratio at which a population of individuals grows. The parameter r is the rate of maximum population growth (i.e. the rate at which the individuals can reproduce)[5] and it is usually considered a positive value. The parameter K represents the maximum number of individuals the environment can sustain. The general solution to the equation can be obtained by separation of variables:

$$\phi^t(y_0) = \frac{K}{1 + (\frac{K}{y_0} - 1)e^{-r(t-t_0)}} \quad (2.5)$$

Where y_0 is the value of $y(t_0)$ in the related IVP.

The system presents two steady points, one (the trivial one) at 0 and the second at K . For what concerns positive values of r , the trivial steady point is unstable while the second one is stable.

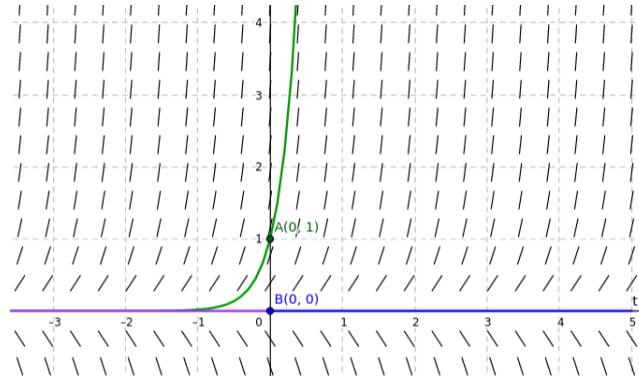


Figure 2.1: The trajectory diagram of the malthus equation, with $r = 4$, showing two solutions.

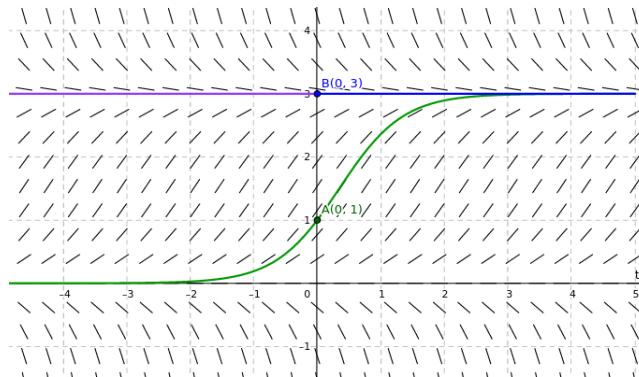


Figure 2.2: The trajectory diagram of the logistic equation, with $K = 3$ and $r = 2$ showing two solutions.

The two trajectory diagrams in Figure 2.1 and 2.2 are numerically obtained. They show two solutions of the respective (2.3) and (2.4) initial

value problems each, with $y(t_0 = 0) = 1$ and $y(t_0 = 0) = 0$ for the Malthus equation and $y(t_0 = 0) = 1$ and $y(t_0 = 0) = 1$ for the Logistic equation. The geometric interpretation allows to qualitatively study IVPs without analytically solving them, giving insights on the problem [4]. Two other fundamental concepts are the orbit of a solution of an ODE and the phase portrait. An orbit in an interval (of the independent variable) can be defined as a subset of the state space such that each value is assumed by the solution in the interval (i.e. $(y(t)) : t \in I$) [4]. On the other hand the phase portrait is a partitioning of the state space into orbits [2]. For what concerns systems of autonomous ordinary differential equations usually also the *phase* or *orbit* plane is presented, this plane doesn't present the independent variable axis (e.g. the time axis), allowing to study the dynamics of a system independently from the time perspective (i.e. study the orbits). The phase diagram can also be plotted for an autonomous ODE, but this will result in a one dimensional line.

The phase plane of a given systems (with proper initial conditions) can present curves (i.e. single orbits) that show some interesting properties, for example closed curves and single points denote special orbits, respectively referred to as a *periodic orbit* and a *steady state* or an *equilibrium*. In Chapter 2.4 the related concepts of stability and bifurcation will be briefly discussed.

2.3 Delay differential equations

Delay differential equations can be seen as a generalization of ODEs and in the mathematical modelling context are crucial to describe certain phenomena not representable by ODEs. With reference to [6] a delay equation is a rule for extending a function of time towards the future on the basis of the (assumed to be) known history. Let $y_t(\theta) := y(t + \theta)$ with $\theta \in [-\tau, 0]$ be the history of the unknown function, which also represents the state, a delay equation can then be defined as $y'(t) = F(y_t)$, $t \geq 0$ where the state space is $X = C([-\tau, 0]; \mathbb{R}^d)$. Note that state y_t is an infinite-dimensional object (a function) [6].

For simplicity only finite delays will be hereby considered, i.e. the function F may depend only on a finite segment of the past history of the unknown function [6].

In order to find a specific solution to a first order delay equation differently from what happens with an ODE, it's not sufficient to specify a single initial value, but an entire function on the interval $[-\tau, 0]$ where τ is the maximum delay in the equation is required. An IVP will then result in a

system resembling:

$$\begin{cases} y'(t) = F(y_t), & t \geq 0 \\ y(\theta) = \psi(\theta), & \theta \in [-\tau, 0] \end{cases}. \quad (2.6)$$

For what concerns the IVP, the theorem of existence and uniqueness of the solution is still valid. A first useful distinction, that will be helpful in Chapter 4, is distinguishing between discrete and distributed delays. The first kind of delay takes the form of the unknown function at time t minus some positive delay term and as such they refer to values of the unknown function at single points in the history of the function itself (e.g. $y(t - \tau_1)$), while distributed delays refer to an interval of values of the function and they are represented by the unknown function inside an integral with the delay term being the integration variable (e.g. $\int_{-\tau}^0 y(t + \theta) d\theta$).

For example the following equation, where τ_1 is a parameter, presents two discrete delays and $\tau = \max\{\tau_1, 1\}$:

$$z'(t) = z(t - 1) + z(t - \tau_1) \quad (2.7)$$

2.4 Stability and bifurcations

The idea of stability of a solution (given an initial point) is both rigorously and intuitively described in [4], it can be intuitively defined as: the property that makes solutions near to the considered solution remain relatively close as the system evolves. On the other hand, if a solution isn't stable it is said to be unstable. Another related concept is the asymptotic stability of a solution, which can intuitively be described as the characteristic such that as time goes to infinity, the distance between the solution and another one obtained by starting with another (relatively close) initial point goes to zero.

When taking into consideration systems depending on one or more parameters such as (2.3) it can be studied how changes in the value of a single parameter affect the solution of the system itself. These systems when described by first order ODEs can be represented by the generic equation:

$$y'(t) = F(p, y(t)), \quad t \geq 0 \quad (2.8)$$

$p \in \mathbb{R}^m$ represents the vector of the m parameters presented by $y(t)$, only real value parameters are being considered in this thesis.

The graph showing how changes in the parameter change (qualitatively) the behavior of the solutions is the *bifurcation diagram* [4] and it can be represented as a Euclidean plane presenting on an axis the parameter chosen for the bifurcation analysis and on the other axis the coordinates for the orbit diagram of the system.

A bifurcation could be described as the values of a considered parameter at which changes in the value of the parameter correspond to qualitative changes in the behavior of the solution. Depending on which aspect of the system's solution changes, bifurcation points can be classified into different classes as described in [2]. An example may be the Hopf bifurcation: A value of the considered parameter at which one equilibrium changes stability and a limit cycle⁴ arises.

2.5 From DDEs back to ODEs

The first step to discretize a delay equation to a system of ODEs is to rewrite a delay differential equation as an abstract differential equation. The generic IVP 2.6 is equivalent to the following abstract IVP [7]:

$$\begin{cases} v'(t) = A_F(v(t)), & t \geq 0 \\ v(0) = \psi, \end{cases} . \quad (2.9)$$

where A_F is the generator associated to the semigroup generated by the solution operators ϕ parametrized in ψ .

Given an IVP, the solution operators ϕ parametrized in ψ means the operators are not dependent on a single initial state x_0 (i.e. the state at time t is described by $\phi^t(x_0)$) but on a whole function ψ (i.e. the state at time t will depend on the parameter ψ , and will be $\phi^t(\psi)$). $A_F(\psi)$ can be defined as the limit as time goes to zero of the difference of evolution operator ϕ considered at the time variable of the limit and ψ , divided by the time variable. $A_F(\psi)$ is then by definition ψ' . Consider now $M \in \mathbb{N} \setminus \{0\}$ distinct points in $[-\theta, 0]$ and a last point equal to 0, for example the $M + 1$ Chebyshev extrema⁵ in $[-\theta, 0]$ and let d be the dimension of the original system. The idea is to approximate the state to a finite dimensional object by representing it through the polynomial interpolation on the nodes. The space $X_M = (R^{d(M+1)})$ is the discretization of the state space X of the original system of index M , every state $x \in X$ is now represented by a vector $x_M \in X_M$ with the components of the vector denoting the state of each of the d equations at the instant specified by the corresponding node. The reconstruction operator R_M associates to a given vector in X_M the M -degree polynomial in X interpolating the values of the vector in the respective considered points. The approximated state will then be described by:

$$R_M(x_M)(\theta) = \sum_{i=0}^M \ell_{M,i}(\theta) x_{M,i}, \quad \theta \in [-\tau, 0]$$

⁴An isolated and closed curve which other orbits approach to or are repelled by it as time progresses

⁵real values in $[-1, 1]$ which can be calculated as $x_k = \cos(\frac{k}{n}\pi)$, $k \in [1, n]$ where x_k is the k -th extrema and n is the number of nodes needed.

Where $x_{M,i}$ is the i -th component of x_M and $\ell_{i,M}$ [7] are the Lagrange polynomials considered on the $M+1$ points chosen, i.e.:

$$\ell_{M,i}(\theta) = \prod_{k=0, k \neq i}^M \frac{\theta - \theta_{M,k}}{\theta_{M,i} - \theta_{M,k}}, \quad i = 0, 1, \dots, M$$

Then F_M , defined as $F(R_M(x_M))$ with $x_M \in X_M$, approximates F . The corresponding discretization of A_F of index M is the operator $A_{F,M}$ defined as $A_{F,M}(y_M) = (F_M(x_M), \Upsilon_M)$ where $\Upsilon_{M,i} = \frac{d}{d\theta_i} R_M(x_M)(\theta_i)$ for $i = 1 \dots M$. The abstract differential equation can be seen as an ODE on X_M . The abstract IVP (2.9) can be turned into:

$$\begin{cases} \frac{d}{dt}(v_M(t)) = A_{F,M}(v_M(t)), & t \geq 0 \\ v_M(0) = x_M, \end{cases} . \quad (2.10)$$

The initial IVP is approximated by an ODE IVP of dimension $d \cdot (M + 1)$ equations.

Why discretizing to an ODE system

There are two main reasons behind the discretization of a dynamical system described by delay equations to an ODE system:

- We are only able to study with the help of a computer finite dimensional objects and it wouldn't be possible to deal with the original problem since the operator A_F mentioned in the previous paragraph is infinite-dimensional.
- While a few software to carry out bifurcation analysis of delayed equations exist (for example DDEBiftool which can treat DDEs with discrete and state dependent delays), softwares for the study of ODE systems are generally more complete and the methods utilized more known.

The presented technique allows also to deal with distributed delays making possible for the user to easily study different kinds of systems.

3

A brief introduction to MatCont

MatCont is a software, based upon MATLAB, that provides functions and a graphic user interface to study the behavior of continuous dynamical systems defined by ODEs, particularly for what concerns the bifurcation analysis[8]. The GUI and functions provided are independent, since the functionalities of MatCont can also be used from MatLab's command line.

In the following paragraphs the main aspects of MatCont's GUI and how the user could interact with it will be discussed, they should be kept in consideration when the extension part will be described in the next chapter. There will not be an exhaustive description of the parsing done by MatCont on the data inserted by the user since during the extension process new parsing functions were introduced without interfering with the pre-existing parsing.

3.1 Overview of the current MatCont GUI

3.1.1 MatCont Main Window

MatCont presents to the user a minimalist but rather intuitive interface. The main window shown to the user is divided into three sections: *Class*, *Current System* and *Current Curve*. *Class* shows the user what type of system is currently loaded (at the moment only ODE), while *Current System* shows the user some basic information about the system itself, such as the name and how the derivatives will be computed (i.e. N for numerically, S for symbolically, and R from window). *Current Curve* shows the curve loaded and what is going to be computed, to give the user some recall of what the state of the software is. In this section, information such as the properties of the initial point (i.e. “initial point type”) and the method that's going to be executed when computing the curve (i.e. “initializer”) are displayed. Above these three sections a toolbar menu is presented, from here the user can create a new system, load, edit or delete an already existing one. The type of the specified initial point can also be selected from this menu (e.g. a point without special properties, an equilibrium etc...) and the same is true for the curve to compute (e.g. an orbit, an equilibrium etc...). The user from this toolbar menu can also open an output window, both numerical and graphical (i.e. a Euclidean plane with two or three axes). The graphical output window will plot the graph (along the specified axes) of what is being computed, at the same time the numerical output window will show

the values of the specified parameters, coordinates and other quantities of interest at every step of the computation.

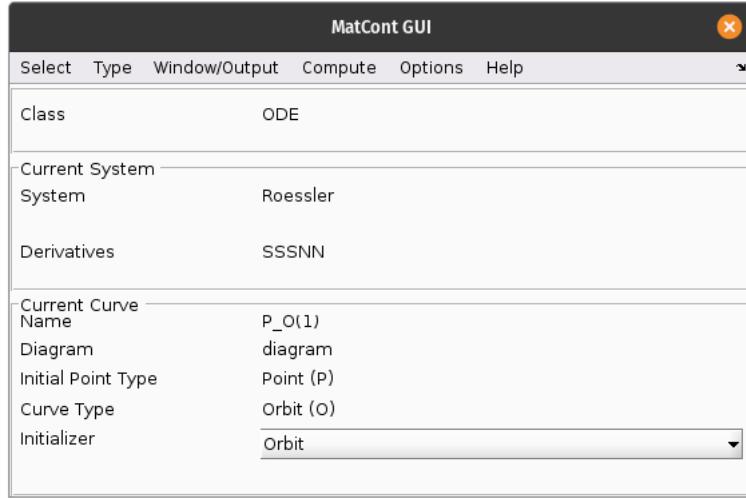


Figure 3.1: MatCont's main window

3.1.2 Inserting a new system

By pressing on Select, System, New in the toolbar menu the user can have access to a new window called *System* from which MatCont allows the user to insert a new system.

This window presents the user several input fields to be filled in:

- Name: Allows the user to specify the name of the system.
- Coordinates: Must contain every coordinate of the system.
- Parameters: Must contain all the parameters used in the system.
- Time: Must contain the name of the independent variable on which the equations in the system depend.

The name of the system must not contain spaces, while the coordinates and the parameters have to be separated by commas.

There is then a table with radio buttons that specify for each order of derivative how it should be computed, *Numerically* the value of the derivative will be numerically approximated, *From window* will allow the user to insert the derivative for each equation in the system (this can be done to save time during the computation) and *Symbolically* which will specify to compute the derivative symbolically (only if the Symbolic toolbox is installed in MatLab). At the bottom of this window a larger text entry window is shown to allow

the user to type the system in. The syntax a system must satisfy requires writing every equation such that the left hand side presents just the first derivative of the coordinate considered. The time dependence is omitted and every operator must be explicitly written.

What happens next? Let for example the name of the new system inserted be “NewTypedSystem”, when the user has correctly inserted all the required data and pressed the *Ok* button MatCont will parse the data entered and create two files under the folder *System* named “NewTypedSystem.m” and “NewTypedSystem.mat” [8]. These files are respectively a MatLab script file, and a binary file. The latter will contain a struct with 36 fields describing the system, such as the list of coordinates, the list of parameters, the equations in the system, the dimension of the system (i.e. the number of different coordinates) etc...

This binary file contains the struct that will be loaded in order to initialize the fields of the current struct representing the system when “Edit” or “Load” is pressed. The .mat file, on the other hand, contains functions such as *fun_eval* and *init* that will be called by MatCont when performing different computations on the system.

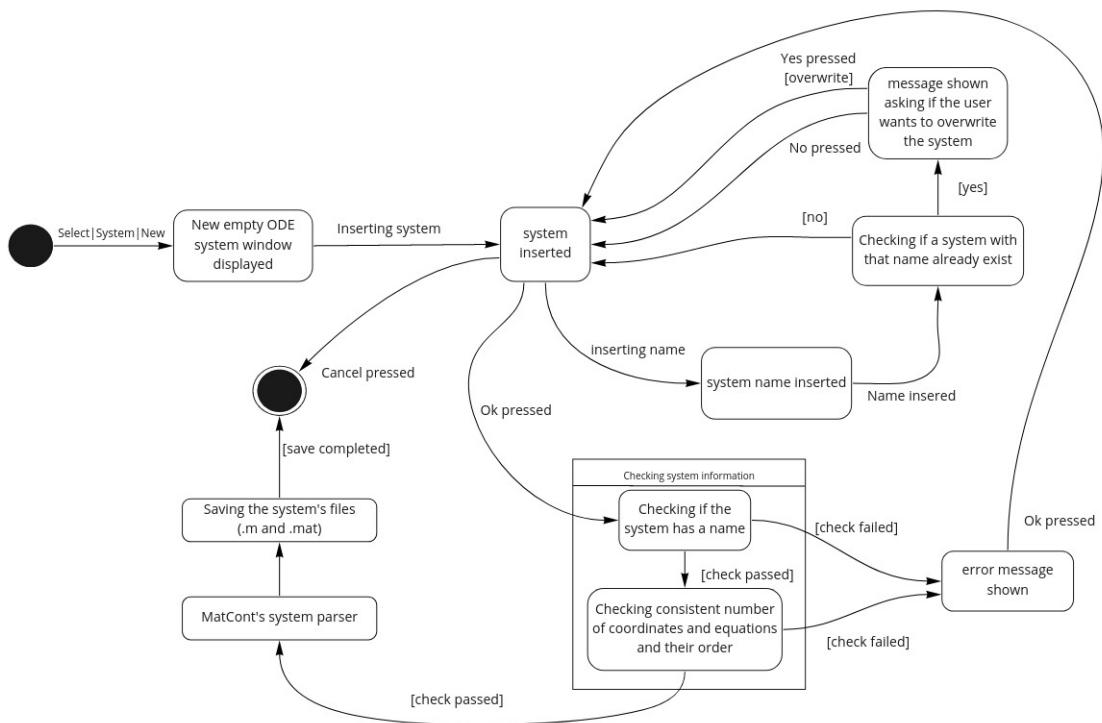


Figure 3.2: The state diagram regarding the insertion of a new system

3.1.3 Editing an existing system

By pressing on *Select, System, Load/Edit/Delete System* the user will be presented a new window in which the list of existing systems and a list of buttons including *Load, Delete* or *Edit* are shown.

When the desired system is selected and “Edit” is pressed the same window used to input the new system is shown, this time though, the .m file related to the system will be loaded and all the fields will be already filled with the system details. The user can then modify what needed and press the *ok* button to confirm the changes made, now both the .m and .mat file will be overwritten with the new information and the system will be reloaded from them. Observe that all the previously computed and saved curves will be deleted by this process.

3.2 The existing documentation

On the page from which MatCont can be downloaded it is possible to find some guides and tutorials that introduce MatCont to a new user¹ by presenting a series of examples that can be repeated step by step. A manual also exists that provides more knowledge on how this software works and how it is structured².

On the other hand the code would considerably benefit from the insertion of comments or method contracts, especially in some regions of the code, which may be quite difficult to read and subsequently modify.

¹<https://sourceforge.net/projects/matcont/files/Documentation/MatContODE/Tutorials7px/>

²<https://sourceforge.net/projects/matcont/files/Documentation/MatContODE/ManualAug2019.pdf>

Extending the user interface of MatCont

In this chapter will be presented how the interface of MatCont has been extended, how this task has been achieved, some difficulties faced and what are now the software capabilities.

4.1 MatCont's extended capabilities

The capabilities of the extended MatCont version (i.e. what systems of differential equations can recognise) have been gradually increased since an iterative development process was used, see section 4.5. The systems that can be studied with MatCont now include all the previous ODE systems (referred in later paragraphs as *standard ODE systems*) as well as DDE systems with one or more discrete delays, such as the Mackey-Glass equation, $y'(t) = \beta \cdot y(t-\tau)/(1+y(t-\tau)^n) - \gamma \cdot y(t)$ and systems with (finite) distributed delays as for example $y'(t) = \int_{-1}^0 k \cdot y(t+z) dz$. Both the integration extremes in distributed delays must fall within one of the following classes:

- A constant (e.g. $\int_{-3}^0 y[t+z] dz$).
- A parameter of the system (e.g. $\int_{-parameter}^0 y[t+z] dz$).
- An expression beginning with the time variable followed by another expression that can contain both constants, parameters and the time variable. Still, the coordinates in the integral must present a delay written as *Coordinate[timeVariable – expression]*.

During the development process some choices were made to allow the extended system to further be expanded easily in order to accept other families of delay equations, such as state dependent DDEs and renewal equations (REs).

REs were indeed supported in later development stages, allowing the system inserted to contain zero or more DDEs and ODEs alongside zero or more REs. All the REs must be specified after the DDEs or ODEs, since the evaluation of the right hand side of each equation is subdivided into two macro blocks containing DDEs/ODEs alongside REs. These later development stages regarding the introduction of REs in the software won't be treated in this thesis.

4.2 Overview of the extended MatCont GUI

From the end user perspective the differences with the original version of MatCont GUI can be found when inserting or editing a system. These differences were intended to be both as localized as possible and consistent with MatCont's GUI style. The latter point has been achieved, the first one (i.e. keeping the differences localized in few specific areas) has been satisfied only from the point of view of the end user.

The latter sentence is referred to the fact that while the graphical differences in the interface can be circumscribed in two main windows, the logic and behavior of MatCont was modified in several different places (i.e. more than a few files present now different behavior accordingly to the type of system considered). This has been proven necessary to be able to fully support the usage of existing MatCont's components and functionalities with the new extension. This idea is fully expressed by Figure 4.1.

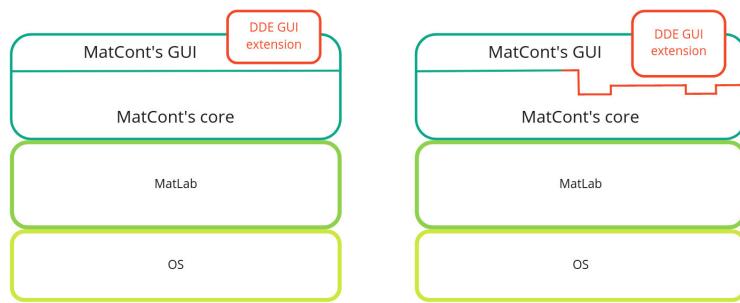


Figure 4.1: Comparison between the initial idea and the implementation.

4.2.1 The DDE syntax

How the user can insert a DDE equation by typing it with natural syntax (i.e. a syntax similar to how the user would write the system with pen and paper) was a major issue during the extension process, since the main goal was to create a user-friendly interface to study DDE systems and if the syntax would have resulted complex and unnatural, the goal wouldn't have been fully achieved.

Creating a method that was able to recognise the syntax was a process carried out during the first phase of the development and extended later on. It largely involved the usage of regular grammars¹.

An equation like $y'(t) = r \cdot y(t)(1 - y(t - \tau))$ could be written in the appropriate window (3.1.2) by typing `y'=r*y*(1-y[t-TAU])` where TAU is

¹A formal grammar whose expressive power corresponds to the expressive power of finite automata

a parameter representing τ (since only ASCII characters are allowed).

In later development stages distributed delay differential equations were added to the equations supported by the software.

An equation like $y'(t) = r \cdot \int_{-\tau}^0 -y(t - \text{DELAY}) d\text{DELAY}$ can be inserted by typing `y' = r * \int_{-\text{TAU}}^0 {-y[t-delay]} \{delay\}`. The syntax used to insert integrals is very similar to the syntax used in L^AT_EX. The `\int_` string marks the beginning of the integral, then the first pair of curly braces preceded by the underscore character contains the inferior integration extreme, the second pair of braces preceded by “`^`” contain the superior integration extreme, then the remaining two pair of braces delimit respectively the function to integrate and the integration variable.

As it can be seen from the examples above the dependencies of the coordinates on specific points in time must be specified with square brackets and contain as the first term of the expression the time variable, e.g. `y[t-1]` or `y[t- TAU]` (where in this latter case TAU is a parameter) followed by an expression describing the delay.

4.2.2 Inserting a new ODE or DDE system

By proceeding with the insertion of a new system in the extended version of MatCont the user will be presented the window in Figure 4.2. From this figure we can observe that the interface did maintain its overall structure, indeed it would have been counterintuitive to create a new interface from scratch, mainly for two reasons, the interface resulted to be already well structured and it would have been harder for users that had already used MatCont for years to adapt to a different interface.

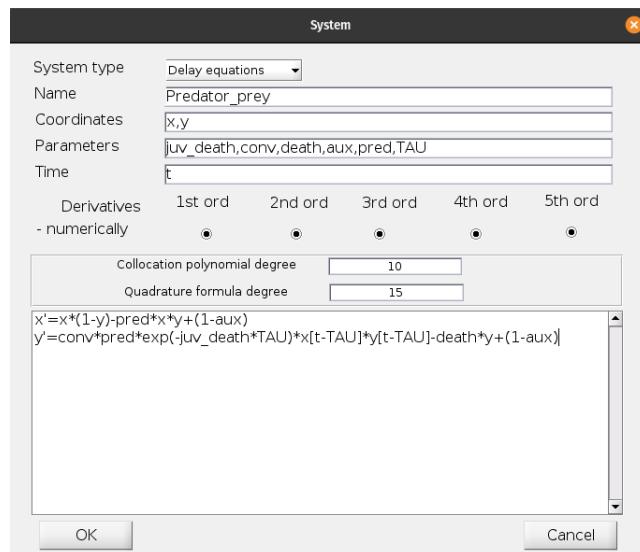


Figure 4.2: MatCont’s window to insert a DDE

From the end user perspective only a few lines of the layout have changed, in the first line line, a dropdown menu lets the user select the type of system to be inserted. If the option “ODE” is chosen then nothing except this first line will be changed, on the other end if “Delay equation” is chosen a subwindow asking the user to input two additional parameters (the degree of the collocation polynomial, and the degree used for the polynomial used for the approximation of integrals) will appear. All the parameters will still be typed in the parameters entry whether they are used to calculate a delay or not. This latter functionality has been modified in later development stages since originally the parameters showing up in the delays (e.g. τ in 4.2.1) had to be specified in an additional entry in the subwindow, alongside the two degrees.

The second change the user can observe is that when inserting a system presenting delay equations there is no other option to compute the derivatives other than numerically. Indeed if the user inserted n equations in the appropriate window the system would have n equations from the end user perspective but, internally it will be represented by $n * (M + 1)$ equations, where M is the degree of the collocation polynomial. If the user could manually specify the derivatives it would need to write $n * (M + 1)$ equations and not just n , breaking the abstraction of the discretization process implemented.

What the user can not see, at least from the graphical interface, is that the binary file associated to a system contains now three additional fields, one describing the type of the system and the other containing, if the system contains delay differential equations, the degree of the collocation polynomial and the degree related to the Clenshaw-Curtis quadrature. The script file containing the functions used by MatCont when studying the system, if it’s related to a “Delay equation” system, is also substantially different when comparing it to the same file associated with an ODE system. The key idea used was based on the same principle adopted when refactoring a function: the methods in the script were semantically modified while keeping the same signature as their ODE counterparts. By doing so it hasn’t been proven necessary to modify every part of MatCont invoking those methods.

4.2.3 Editing an ODE or DDE system

The view the user is presented when a system to edit is selected it is still the same of when a new system is inserted, with the exception that now, the dropdown menu that would allow the user to select the system type is locked on the type of the system that is being modified.

When the system considered is a standard ODE system, meaning a system previously inserted with the original version of MatCont, or an ODE system inserted with the extended version of MatCont, the user will be presented the same system details as in the original MatCont that could be modified

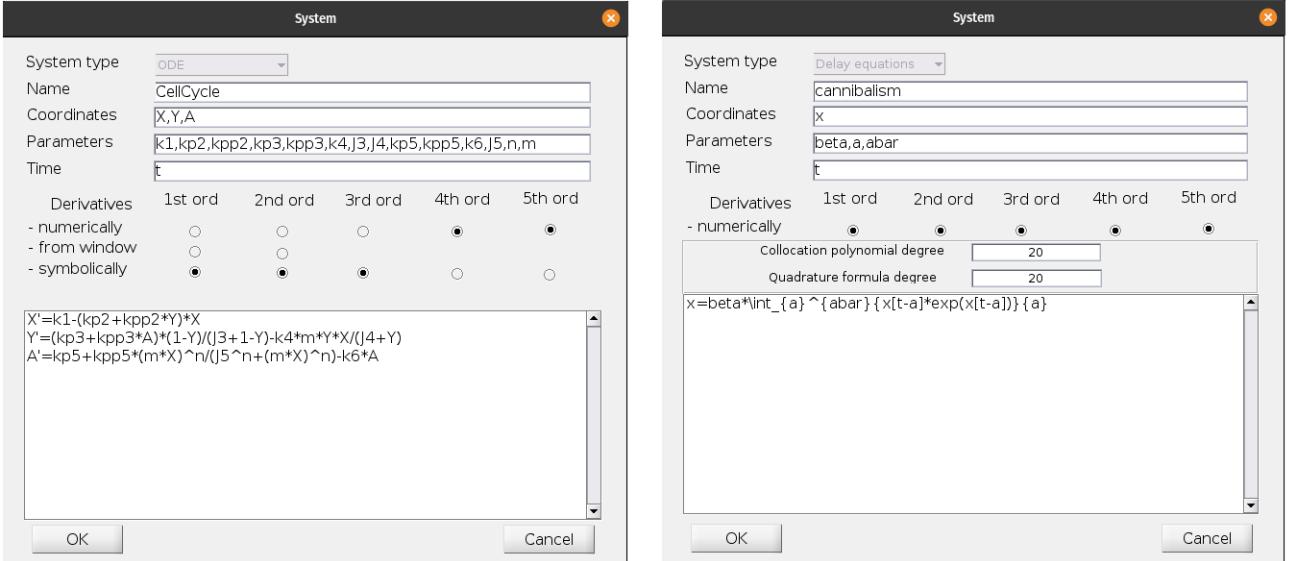


Figure 4.3: The comparison between MatCont’s window to edit an “ODE” and “Delay equation” system.

(i.e. the name, the coordinates, the parameters, the time, and the equations themselves).

On the other hand, if the system considered contained delay equations, the interface will display (as when the insertion of a new “Delay equation” system is performed) the subpanel containing the two entries with the degree both the the collocation polynomial and quadrature formula chosen. These values can be modified alongside all the details discussed above for the “ODE” systems.

4.3 The complete legacy support

As written in 4.2.2 the binary file (.mat) associated to a system contains now some additional fields, this could make someone question whether the extended version of MatCont will still work with systems (and thus files) created with the regular version of MatCont (legacy systems).

One of the main goals was indeed to achieve full compatibility with these systems, and this is mainly done when a system is loaded. When MatCont now loads a file checks whether the field denoting the system type in the binary file exists or not. If it finds to be the latter case then the value of the field loaded is set to “ODE”, since in regular MatCont only these types of systems are allowed (the struct in which the data is loaded contains indeed all the fields, regardless of what structure is saved on the file). From this point in the code everything will work the same as for ODE systems inserted

with the extended version of the software.

This compatibility with previously created systems can easily be understood from Figure 4.4 which describes the states the extended MatCont can go through when inserting the new system. Comparing it with Figure 3.2 the new states are represented by a green rectangle, and it can be observed that the overall structure remains similar while special states for managing the new cases are added. This modus operandi lead also the extension for the other few MatCont's components involved.

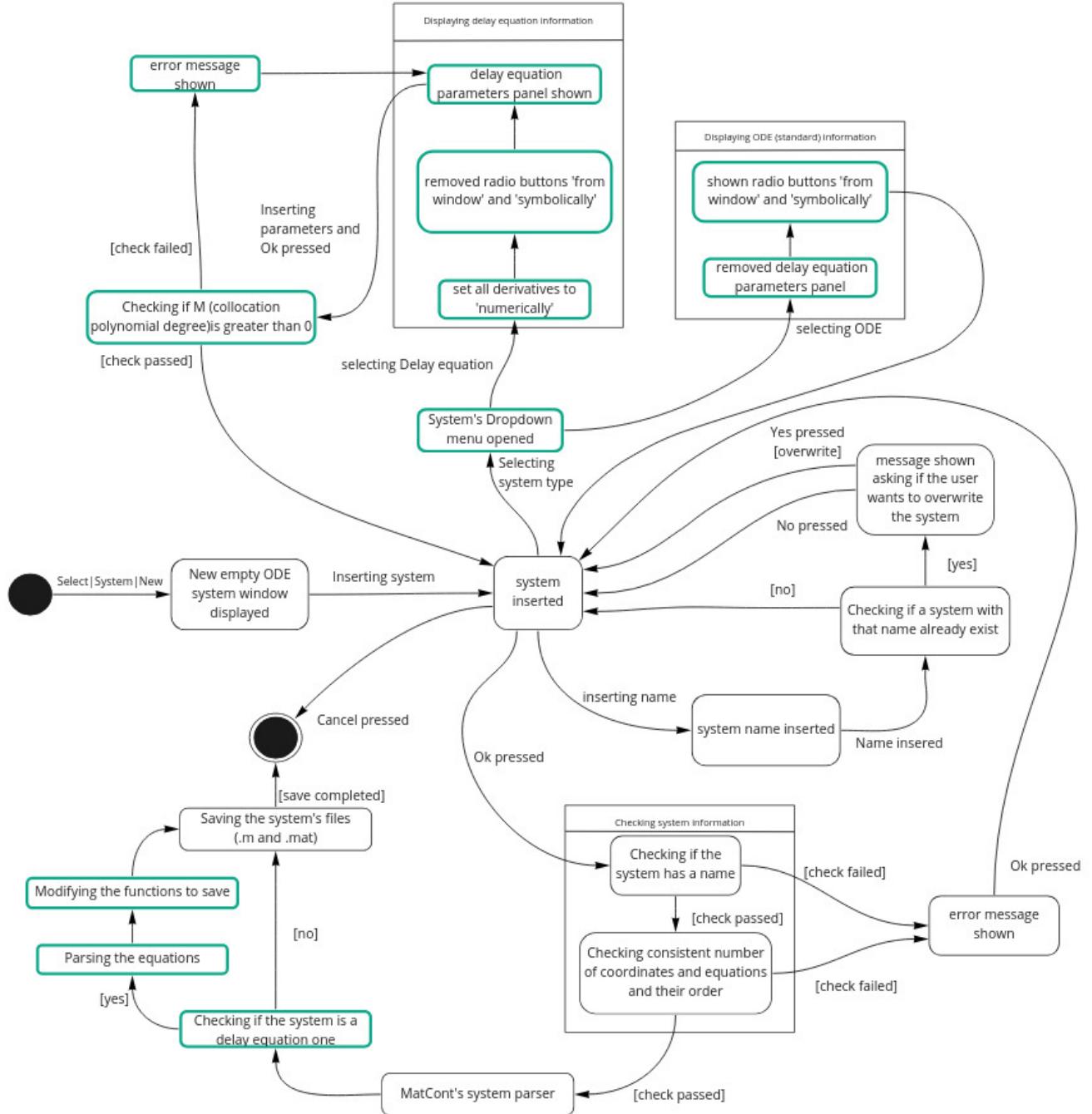


Figure 4.4: The state diagram regarding the insertion of a new system in the extended MatCont. The new states are marked by the green border.

4.4 A deeper view of the parsing process

The additional parsing process on the equations of the system, as can be observed from Figure 4.4, takes place only in the case of “Delay equation” systems, since otherwise the system won’t contain neither discrete nor distributed delays.

This parsing phase is responsible for translating the equations in the system written by the user to the format the system will present in its discretized form in the *fun_eval* function in the relative .m file. The *fun_eval* function returns a vector containing the evaluation of the system with the specified state and value of the parameters.

The parsing process consist of two main phases, a first routine where each equation of the system is checked for delayed terms (since some equations in a “Delay equation” system may not contain delays) with a regular expression like the one presented below:

```
"coordinateVar \[ timeVar \W ( \[ timeVar \W [^\\]] * \\] | [^\\]])) * \\]"
```

The meaning of the regular expression above applied on an equation of the system can be interpreted as: “Find in the right hand side of the equation the substrings starting with the specific coordinate, followed by a open square bracket and the time variable, followed by a non alphabetic nor numeric symbol² (expecting the user to insert a + or -) followed either by another open square bracket, another non alphabetic nor numeric symbol and a string not containing the close square bracket, or followed by just the latter string. Then repeat this pattern for zero or more times (Kleene star operator) and lastly the substring must end with a close square bracket”.

By doing this it is then possible to extrapolate each delay and replace the substring with another properly built string that contains a function call to a static method defined inside another script file that is responsible for computing the interpolation.

This process is repeated for each equation with all the different coordinates (since every equation might contain arbitrary delayed terms). The structure of the regular expression above would already allow the recognition of state dependent equations up to one nested level.

The second routine of the parsing phase, also mainly based on the usage of regular expressions, is responsible for recognising and properly substituting integrals that may appear in the right hand side of an equation. It follows the same principle as the first routine, for each equation, with a properly built regular expression, find all the integrals, recognise and substitute the integration variable and eventually substitute the integral in the right hand side with the correct string representing the approximated integral that will

²A character not in the[a-zA-Z_0-9] set.

be evaluated in computing *fun_eval*. These two phases are carried out sequentially, so the string representing the right hand side of each equation in the discretized system produced in output by the first routine is then taken as an input in the second parser routine.

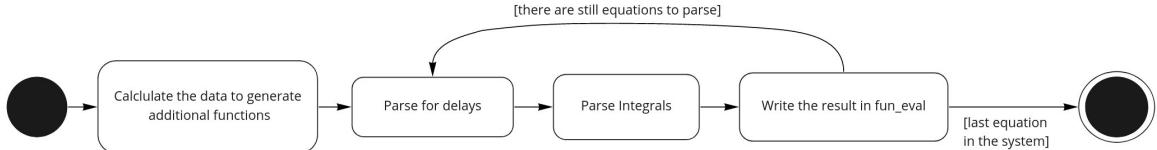


Figure 4.5: A more detailed state diagram of the “parsing the equations” state of Figure 4.4

There are also other secondary less complex routines that write in the .m file other sections of both the *fun_eval*, *init* and other functions.

One of the routines mentioned is responsible for writing in the file some functions each returning a pre-computed matrix. For example one function returns the $M+1^3$ Chebyshev nodes in the interval $[-1, 0]$.

Another one of the secondary routines is responsible for writing in *fun_eval* an array identified by the name “delayFunctions” containing all the delays found in the system. The process used to find the delays is analogous to the one used during the first parsing phase. The array written in *fun_eval* will be used in every evaluation of the system to determine the maximum (without sign) delay and subsequently scale the Chebyshev nodes (used to interpolate the behavior of the solution in the interval $[-\tau_{max}, 0]$) from the interval $[-1, 0]$ to $[-\tau_{max}, 0]$.

³where M is the degree of the collocation polynomial

4.5 A brief look at the development process

The extension process followed some of the development principles presented in the Agile Manifesto [9]. Focus was especially put on the delivery of working software frequently and on the iterative development process. Overviewing the iteration process the evolution of the software went through these major intermediate phases:

- Development of the GUI extension (without any behavior attached).
- Integration of the structure used by MatCont to memorize systems in files and integration of the behavior of the extended interface.
- Adjustment of the preprocessing done on the data inserted by the user before passing it to the numerical algorithms (at this point the numerical part of MatCont, such as integration or the continuation of an equilibrium, was working correctly with DDEs with discrete delays).
- Integration of MatCont's scripts related to the memorization of the computed data (now the part of MatCont related to plotting the computed curve worked correctly with DDE systems).
- Extension of the preprocessing and parsing phase done on the input to recognise distributed delays (i.e. parsing integrals).
- Extension of the preprocessing and parsing phase done on the input to recognise renewal equations.

Each intermediate version was followed by a small testing phase, consisting primarily of Unit testing, a related debugging phase and a code documentation phase. Towards the ending phases of the development some system testing (e.g. higher level testing aiming to check the quality of the whole system [10]) took place. These last tests were carried out with ad-hoc dynamical systems and by creating, studying and comparing the results of some of the examples studied with DDE-BifTool⁴ that can be found on its reference webpage⁵.

During the development two main challenges were faced:

- Working on a software (MatCont) related to a previously unknown domain to me. This situation can be quite common when dealing with softwares in different fields. How this challenge was faced and overcame was by conducting an introductory study on the main aspects of the field and also by interacting and working with a domain expert.

⁴A software for the bifurcation analysis of delay-differential equations

⁵<http://ddebiftool.sourceforge.net/demos/index.html>

- Understanding and extending sections of code with little or none code documentation (i.e. comments or method contracts) as introduced in section 3.2. This challenge was mostly faced when adapting the code responsible for handling the curve data. Overall, this challenge has been faced with the help of some of the tools provided by MatLab for the debugging process such as breakpoints and step-by-step execution.

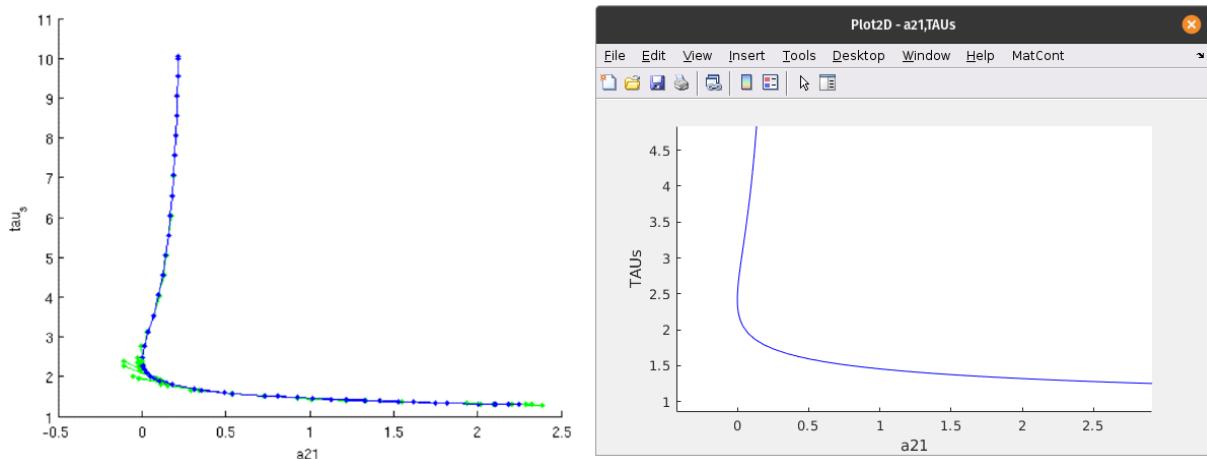


Figure 4.6: A qualitative comparison between a Hopf bifurcation continuation curve of DDE-Biftool and Matcont's.

5

Conclusions

The opportunity of this internship and the work done helped me in gaining a better perspective of how much the job of a computer scientist can vary, while also giving a first glimpse on a scientific field previously unknown to me. Understanding that there is always room for improvement, the software could be further extended in the future, in the next paragraph some of these modifications are presented.

5.1 Future extensions

The system has been extended keeping in mind that other eventual extensions may take place. For example nested integrals or state dependent delay equations (up to some level of nesting) aren't supported yet, but in future these features could be supported and properly managed by a more general extension of MatCont. The most intuitive way to extend the syntax presented in 4.2.1 to allow the insertion of state dependent delay equations would be to let the user write them in a form like the following: $y' = y[y[t - TAU]]$ where this string represents the equation $y(t)' = y(y(t - \tau))$, so by expressing nested delay dependencies also with square brackets.

Another possible future integration more related to the GUI and the User experience itself, may be the automatic recognition of the system type once inserted, by doing so, the user wouldn't need to select whether the system inserted is described by ODEs only, or by delay equations. This will bring the GUI even closer to the standard MatCont's GUI.

Bibliography

- [1] D. K. Arrowsmith, C. M. Place, C. Place, *et al.*, *An introduction to dynamical systems*. Cambridge university press, 1990.
- [2] Y. A. Kuznetsov, *Elements of Applied Bifurcation Theory*. Springer, 1998.
- [3] M. M. Meerschaert, “Chapter 4 - introduction to dynamic models,” in *Mathematical Modeling (Fourth Edition)* (M. M. Meerschaert, ed.), pp. 115–137, Boston: Academic Press, fourth edition ed., 2013.
- [4] C. Chicone, *Ordinary Differential Equations with Applications*. Texts in Applied Mathematics, Springer New York, 2008.
- [5] E. W. Weisstein, “Logistic equation,” <https://mathworld.wolfram.com/>, 2003.
- [6] O. Diekmann, S. A. Van Gils, S. M. Lunel, and H.-O. Walther, *Delay equations: functional-, complex-, and nonlinear analysis*, vol. 110. Springer Science & Business Media, 2012.
- [7] D. Breda, O. Diekmann, M. Gyllenberg, F. Scarabel, and R. Vermiglio, “Pseudospectral discretization of nonlinear delay equations: New prospects for numerical bifurcation analysis,” *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 1–23, 2016.
- [8] W. Govaerts, Y. A. Kuznetsov, H. Meijer, B. Al-Hdaibat, V. De Witte, A. Dhooge, W. Mestrom, N. Neirynck, A. Riet, and B. Sautois, “Matcont: Continuation toolbox for odes in matlab,” *U. Gent*, 2018.
- [9] M. Fowler, J. Highsmith, *et al.*, “The agile manifesto,” *Software development*, vol. 9, no. 8, pp. 28–35, 2001.
- [10] L. Luo, “Software testing techniques,” *Institute for software research international Carnegie mellon university Pittsburgh, PA*, vol. 15232, no. 1-19, p. 19, 2001.